Estudio Técnico del Proyecto

Proyecto: ByteBuss

Equipo de Desarrollo: Los pozoles

Caro

Raúl

Zacek

Edgar

Estudio Técnico del Proyecto

Introducción

ByteBuss es un proyecto tecnológico integral para la **gestión inteligente del transporte público urbano**.

Su **estudio técnico** detalla la **arquitectura del sistema**, **tecnologías elegidas**, **patrones de diseño aplicados**, **estrategia de pruebas**, y **herramientas de soporte**, garantizando **eficiencia, calidad y mantenibilidad**.

Arquitectura Propuesta

El sistema se basa en **arquitectura Modelo-Vista-Controlador (MVC)** para **separar responsabilidades**, mejorar **mantenibilidad** y **facilitar la colaboración en equipo**.

🔹 Ventajas de la arquitectura MVC

- Separación clara entre lógica de negocio, datos y presentación.
- Facilita el mantenimiento y escalabilidad.
- Compatible con frameworks modernos.
- Mejora la reutilización de componentes.

🔹 Distribución de Capas

Modelo

- Entidades: Usuarios, Rutas, Autobuses, Boletos.
- Lógica de negocio y validaciones.
- Conexión con bases de datos (MongoDB / PostgreSQL).

Vista

- Interfaces responsivas (Web y App Móvil).
- Panel administrativo para operadores.

- App para conductores y usuarios finales.

Controlador

- Recepción de solicitudes.
- Procesamiento de lógica.
- Exposición de APIs RESTful.

Frameworks y Tecnologías

Componente	**Tecnologías Seleccionadas**
Backend	Java Spring Boot / Node.js (Express).
Frontend	React (Web), Kotlin / React Native (Móvil).
Base de Datos	MongoDB o PostgreSQL según módulo.
Infraestructura	Docker, Nginx, AWS/Azure para despliegue en la nube.
CI/CD	GitHub Actions.
Control de versiones	GitHub con estrategia Git Flow.

Patrones de Diseño Aplicados

Patrón	**Uso en ByteBuss**	**Ventajas**
Singleton	Conexión única a la base de datos o servicios externos.	Evita instancias múltiples, ahorro de recursos.
Repository	Abstracción del acceso a datos.	Facilita mantenimiento y pruebas.
Factory	Creación de objetos complejos (Usuarios con roles).	Centraliza lógica de instanciación, mejora extensibilidad.
Observer	Notificaciones en tiempo real de ubicación de autobuses.	Bajo acoplamiento, actualizaciones automáticas.
Façade	Simplifica acceso a subsistemas (ventas, pagos).	Interfaz unificada, oculta complejidad interna.

Controller	Gestión de peticiones en capa de API.	Flujo claro, organización del código.

Justificación Técnica

- **MVC:** Organiza el desarrollo en capas, facilita mantenimiento.
- **React / Kotlin:** Interfaces modernas y responsivas.
- **Spring Boot / Node.js:** APIs robustas y escalables.
- **MongoDB / PostgreSQL:** Flexibilidad y rendimiento en almacenamiento.
- **Docker / Nginx:** Contenedores reproducibles y balanceo de carga.
- **GitHub + Actions:** Control de versiones con CI/CD integrado.

Responde a necesidades de **modularidad, escalabilidad y calidad**.

Estrategia de Pruebas

Tipo de Prueba	**Objetivo**	**Responsable**	**Herramientas Sugeridas**
Pruebas Unitarias	Verificar funciones individuales.	Desarrolladores.	Jest, PyTest.
Pruebas de Integración	Validar comunicación entre módulos.	Todo el equipo.	Postman, Supertest.
Pruebas Funcionales	Verificar casos de uso definidos.	QA / Testers.	Casos de prueba manuales.
Pruebas de UI	Verificar interfaces y usabilidad.	UI/UX Designers.	Navegadores, DevTools.
Pruebas de Aceptación	Validar con el cliente funcionalidades finales.	Equipo + Cliente.	Demos en vivo, Formularios.

Control de Calidad

- Cobertura mínima de pruebas unitarias (>80%).
- Revisión de código vía **Pull Requests** en GitHub.
- Validación de criterios de aceptación por Sprint.
- Auditorías mensuales de seguridad.
- Documentación actualizada en cada entrega.

Herramientas de Soporte

- **Jira/Trello:** Planificación y seguimiento de sprints.
- **Slack/Teams:** Comunicación diaria y ágil.
- **Google Drive/Notion:** Documentación colaborativa.
- **GitHub:** Control de versiones, colaboración.
- **AWS/Azure:** Infraestructura escalable y segura.